Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
P. Bamberger
P. Schneider

Algorithmen und Datenstrukturen
Sommersemester 2020
Musterlösung Übungsblatt 11

Abgabe: Mittwoch, 29.07.2020, 16:00 Uhr.

Aufgabe 1: Bitstrings ohne aufeinanderfolgende Einsen *(10 Punkte)*

Gegeben eine natürliche Zahl $n \geq 1$, möchte man die Anzahl der n-stelligen bitstrings berechnen, welche keine zwei aufeinanderfolgende Einsen enthalten (für $n = 3$ bspw. wäre diese Anzahl 5, da 000, 001, 010, 100, 101 genau die 3-stelligen bitstrings sind, welche keine zwei aufeinanderfolgende Einsen enthalten).

(a) Geben Sie einen Algorithmus an, welcher dieses Problem in $\mathcal{O}(n)$ Zeit löst. Erklären Sie die Laufzeit. *(6 Punkte)*

(b) Implementieren Sie Ihre Lösung. Sie können dazu die Vorlage `DP.py` benutzen. Wenden Sie Ihren Algorithmus auf die Werte 10, 20 und 50 an und schreiben Sie die Ergebnisse in Ihre `erfahrungen.txt`. *(4 Punkte)*

Musterlösung

(a) Sei $A(n, i)$ die Anzahl n-stelliger Bitstrings, welche keine zwei aufeinanderfolgende Einsen enthalten und auf i enden, für $i \in \{0, 1\}$. Wir haben $A(n, 0) = A(n-1, 0) + A(n-1, 1)$ sowie $A(n, 1) = A(n-1, 0)$. Es folgt $A(n, 0) = A(n-1, 0) + A(n-2, 0)$ mit den Basisfällen $A(1, 0) = 1$ und $A(2, 0) = 3$. Die rekursive Struktur ist also die gleiche wie bei den Fibonacci-Zahlen. Berechnung und Laufzeit geht daher analog zur Vorlesung (Woche 11, Folie 6). Die Anzahl n-stelliger Bitstrings ohne aufeinanderfolgende Einsen ist $A(n + 1, 0)$.

(b) Siehe `DP.py` (andere Implementierung als in (a) beschrieben). Die Werte für 10, 20 und 50 lauten 144, 17711 und 32951280099.

Aufgabe 2: Summengleiche Partitionierung *(10 Punkte)*

Gegeben sei eine Menge $X = \{x_0, \ldots, x_{n-1}\}$ mit $x_i \in \mathbb{N}$. Wir möchten bestimmen *ob* es eine Teilmenge $S \subseteq X$ *gibt*, so dass $\sum_{x \in S} x = \sum_{x \in X \setminus S} x$. Es ist nicht nötig S zu berechnen!

(a) Sei $W := \sum_{x \in X} x$. Stellen Sie eine rekursive Formel $s : \{0, \ldots, n\text{–}1\} \times \{0, \ldots, W\} \to \{\texttt{True, False}\}$ auf, mit $s(i, j) = \texttt{True}$ genau dann wenn es eine Teilmenge $S \subseteq \{x_0, \ldots, x_i\}$ gibt mit $\sum_{x \in S} x = j$. Erläutern Sie wie s benutzt werden kann um obiges Problem in $\mathcal{O}(W \cdot n)$ Zeit zu lösen. *(6 Punkte)*

(b) Implementieren Sie Ihre Lösung. Sie können dazu die Vorlage `DP.py` benutzen. Lesen Sie die Mengen `set1.txt`, `set2.txt` und `set3.txt` ein und wenden Sie Ihren Algorithmus darauf an. Schreiben Sie die Ergebnisse in Ihre `erfahrungen.txt` *(4 Punkte)*

Musterlösung

(a) Angenommen es gibt eine Menge $S \subseteq \{x_0, \ldots, x_i\}$ mit $\sum_{x \in S} x = j$. Dann muss entweder $x_i \in S$ sein oder $x_i \notin S$. Im ersten Fall gibt es eine Menge $S' \subseteq \{x_0, \ldots, x_{i-1}\}$ mit $\sum_{x \in S'} x = j - x_i$. Im zweiten Fall gibt es eine Menge $S'' \subseteq \{x_0, \ldots, x_{i-1}\}$ mit $\sum_{x \in S''} x = j$. Falls so eine Menge S nicht existiert, dann gibt es weder S' noch S''. Wir haben also die Äquivalenz, dass S existiert genau dann wenn S' oder S'' existieren. Darauf aufbauend definieren wir rekursiv

$$s(i, j) = s(i - 1, j - x_i) \vee s(i - 1, j)$$

wobei $\vee$ den logischen Oder-Operator darstellt welcher wahr ist falls eine der beiden Argumente wahr ist. Außerdem definieren wir Basisfälle $s(i, 0) =$ `True` (dann ist die leere Menge eine gültige Lösung), $s(0, j) =$ `True` falls $x_0 = j$, sowie $s(i, j) =$ `False` falls $i < 0$ oder $j < 0$.

Falls es eine Menge $S \subseteq X$ gibt mit $\sum_{x \in S} x = \sum_{x \in X \setminus S} x$ dann müssen beide Summen offensichtlich $W/2$ ergeben. Wir erhalten also die Lösung unseres Problems indem wir $s(n-1, W/2)$ berechnen.

Wir wenden dynamisches Programmieren an um $s(n-1, W/2)$ zu berechnen. Da i und j bei der rek. Berechnung nur kleiner werden, haben wir nur $n \cdot (W/2 + 1) = \mathcal{O}(n \cdot W)$ unterschiedliche Möglichkeiten für den Parameterraum (i, j). Wir müssen also $\mathcal{O}(n \cdot W)$ mal den Wert von $s(i, j)$ berechnen.

Die Berechnung eines einzelnen Wertes mittels der rekursiven Formel $s(i, j) = s(i - 1, j - x_i) \vee s(i - 1, j)$ *ohne* die Berechnungskosten durch die Rekursion, ist lediglich $\mathcal{O}(1)$ (Berechnung eines logischen Oders von zwei Bits und ein paar Überprüfungen von Basisfällen und Zuweisungen). Wir speichern alle Zwischenwerte $s(i, j)$ in einem Dictionary `memo[i,j]` und müssen somit jeden Wert $s(i, j)$ nur einmal berechnen. Die Gesamtlaufzeit ist also $\mathcal{O}(n \cdot W)$.

(b) Siehe `DP.py` unserer Musterlösung. Die Ergebnisse angewandt auf die gegebenen Mengen `set1.txt`, `set2.txt` und `set3.txt` sind jeweils `True, False, True`.

Anmerkung: Leider war unsere Probleminstanz zu groß gewählt. Bei `set1.txt` und `set3.txt` ging es dennoch recht schnell weil dies `True`-Instanzen waren und die greedy Auswertung des Oder-Operators einen massiven Speedup bringt. Bei der `False`-Instanz `set2.txt` musste allerdings der ganze Parameterraum $n \cdot (W/2 + 1) \approx 4 \cdot 10^8$ durchsucht werden. Bei meinem Rechner dauerte das knapp 2 Minuten lang.